Group Project - Neural Networks

Naeem Nowrouzi, Sardar Singh, Naby Diop

May 10, 2018

```r
library(Matrix)
library(foreach)
library(ISLR)
library(leaps)
library(CosmoPhotoz)
library(glmnet)
library(deepnet)
library(neuralnet)
library(pls)
library(tree)
library(randomForest)

# The dataset from the CosmoPhotoz package consists of measurements of astronomical
# photometric bands and redshifts.We seek to predict the value of redshift for a given
# galaxy, given the other measurements.

data("PHAT0train")
data("PHAT0test")

# combine observations
phat0=rbind.data.frame(PHAT0train, PHAT0test)
dim(phat0)
```

```
## [1] 169520     12
```

```r
names(phat0)
```

```
##  [1] "redshift" "up"       "gp"       "rp"       "ip"       "zp"
##  [7] "Y"        "J"        "H"        "K"        "IRAC_1"   "IRAC_2"
```

```r
length(names(phat0))
```

```
## [1] 12
```

```r
sum(is.na(phat0)) # no missing data
```

```
## [1] 0
```

```r
# Next, we sample from the larger phat0 dataset to get the training and test sets.

train.amount = 2500
test.amount = 1000

set.seed(1)
train.index = sample(1:nrow(PHAT0train), train.amount)
test.index = sample(1:nrow(PHAT0test), test.amount)

phatzero.train = PHAT0train[train.index,]
phatzero.test = PHAT0test[test.index,]
```

```r
# Fit a simple linear regression on the training set
lm.fit=lm(redshift~., data=phatzero.train)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = redshift ~ ., data = phatzero.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.27105 -0.05455 -0.00499  0.05193  0.80638
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.167698   0.044881  -3.736 0.000191 ***
## up          -0.181200   0.006720 -26.965  < 2e-16 ***
## gp           0.015211   0.015086   1.008 0.313418
## rp           0.306182   0.015065  20.324  < 2e-16 ***
## ip           0.061619   0.022761   2.707 0.006831 **
## zp          -0.066794   0.022309  -2.994 0.002780 **
## Y           -0.073702   0.021636  -3.406 0.000669 ***
## J            0.625942   0.030155  20.758  < 2e-16 ***
## H            0.381004   0.032107  11.867  < 2e-16 ***
## K           -1.047835   0.023119 -45.324  < 2e-16 ***
## IRAC_1      -0.062236   0.003439 -18.099  < 2e-16 ***
## IRAC_2       0.063697   0.003830  16.630  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08725 on 2488 degrees of freedom
## Multiple R-squared:  0.8655, Adjusted R-squared:  0.8649
## F-statistic:  1455 on 11 and 2488 DF,  p-value: < 2.2e-16
```

```r
# We see in the summary of the model that some of the predictors are statistically
# significant. The adjusted R-squared is about 0.86 which suggests that the relationship
# might be linear. The residuals suggest that the normal distribution assumption of the
# model is correct. The residual standard error is quite low at 0.087.

# Compute the test error for linear regression
pred.lm = predict(lm.fit, phatzero.test)
error.lm = mean((phatzero.test$redshift - pred.lm)^2)
error.lm
```

```
## [1] 0.007340993
```

```r
# Fit a Ridge regression on the data
xmatrix.train = model.matrix(redshift~., data=phatzero.train)
xmatrix.test = model.matrix(redshift~., data=phatzero.test)

fit.ridge = cv.glmnet(xmatrix.train, phatzero.train$redshift, alpha=0)

lambda.ridge = fit.ridge$lambda.min

pred.ridge = predict(fit.ridge, s=lambda.ridge, newx=xmatrix.test)
error.ridge = mean((phatzero.test$redshift - pred.ridge)^2)
```

```r
error.ridge
```

```
## [1] 0.01520708
```

```r
# Fit a Lasso model on the trainin data
fit.lasso = cv.glmnet(xmatrix.train, phatzero.train$redshift, alpha=1)
lambda.lasso = fit.lasso$lambda.min

# get the predicted values on the test set
predict.lasso = predict(fit.lasso, s=lambda.lasso, newx=xmatrix.test)

# Compute the test MSE
error.lasso = mean((phatzero.test$redshift - predict.lasso)^2)
error.lasso
```

```
## [1] 0.007304559
```

```r
# Fit a PCR model
set.seed(1)
fit.pcr = pcr(redshift~., data=phatzero.train, scale=TRUE, validation="CV")

# Get the predicted values on the test set
predict.pcr = predict(fit.pcr, phatzero.test, ncomp=length(names(phat0))-1)

# Get the test MSE.
error.pcr = mean((phatzero.test$redshift - predict.pcr)^2)
error.pcr
```

```
## [1] 0.007340993
```

```r
# Fit a Partial Least Square to the data
set.seed(1)
fit.pls=plsr(redshift~., data=phatzero.train, scale=TRUE, validation="CV")

# Get the predicted values on the test set
predict.pls = predict(fit.pls, phatzero.test, ncomp=length(names(phat0))-1)

# Get the test MSE
error.pls = mean((phatzero.test$redshift - predict.pls)^2)
error.pls
```

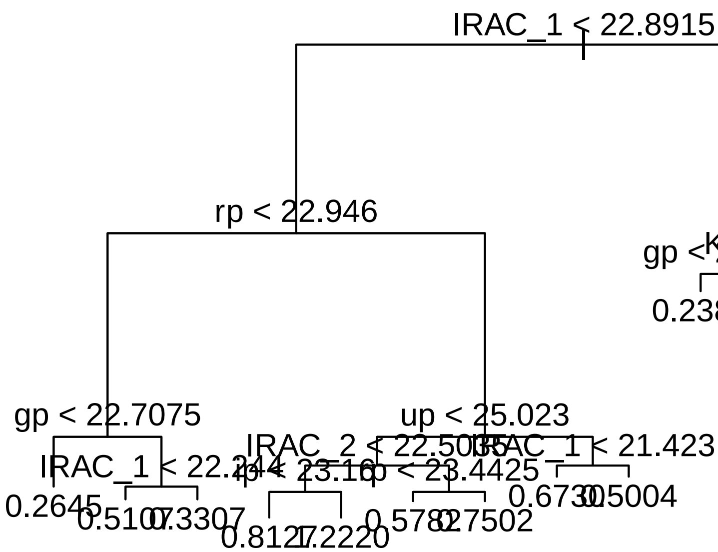
```
## [1] 0.007340993
```

```r
# Regression Trees
set.seed(1)
reg.tree.phat0 = tree(redshift~., data=phat0, subset=train.index)
summary(reg.tree.phat0)
```

```
##
## Regression tree:
## tree(formula = redshift ~ ., data = phat0, subset = train.index)
## Variables actually used in tree construction:
## [1] "IRAC_1" "rp"     "gp"     "up"     "IRAC_2" "ip"     "K"
## Number of terminal nodes:  14
## Residual mean deviance:  0.01726 = 42.9 / 2486
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
```

```
## -0.48480 -0.09278 -0.01300  0.00000  0.08196  0.53550
```

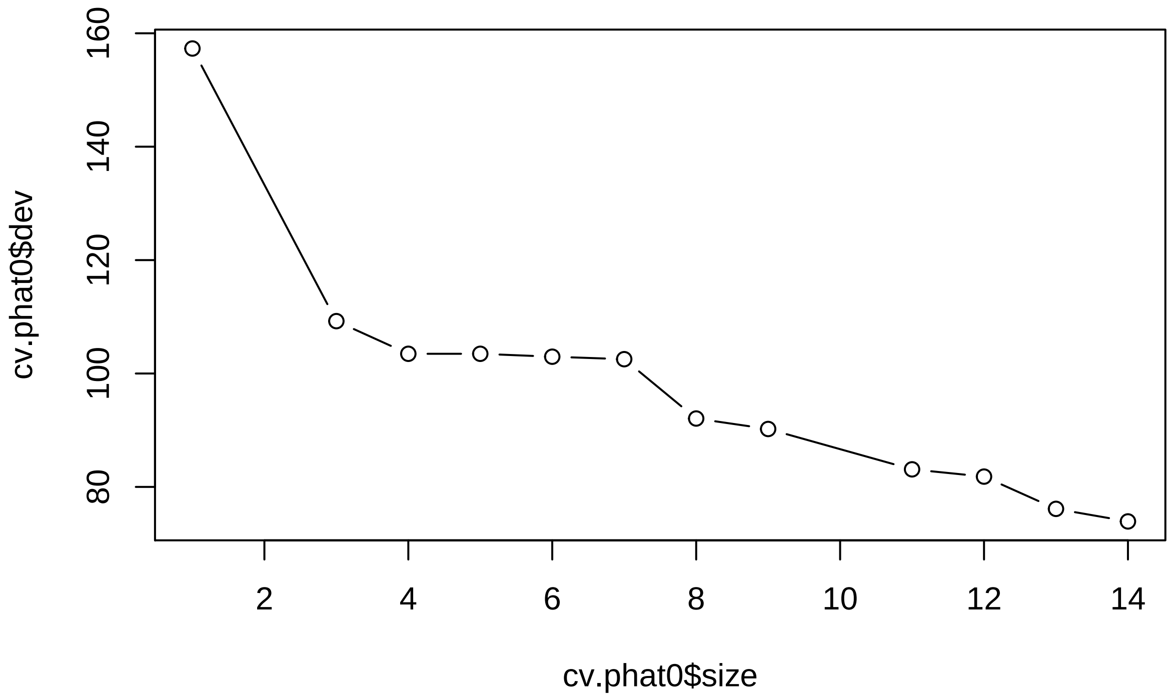
```r
# Plot the tree
plot(reg.tree.phat0)
text(reg.tree.phat0, pretty=0)
```




```r
# The tree has 14 terminal nodes. We should prune the tree, but we shall obtain
# error measurements for both the unpruned and pruned trees.

# error for the unpruned tree
unpruned.pred = predict(reg.tree.phat0, newdata=phatzero.test)
error.unpruned = mean((unpruned.pred - phatzero.test$redshift)^2)
error.unpruned
```

```
## [1] 0.01932991
```

```r
# Determine the oprtimal tree size by cross-validation.
cv.phat0=cv.tree(reg.tree.phat0)
plot(cv.phat0$size, cv.phat0$dev, type="b")
```




```
##Determine the optimal tree size
cv.phat0$size[which.min(cv.phat0$dev)]
```

```
## [1] 14
```

```
# 14 terminal nodes, that is, all of the variables included, gives the lowest
# deviance (or sum of squared errors). Thus we do not need to prune the tree.#

prune.phat0 = prune.tree(reg.tree.phat0, best=cv.phat0$size[which.min(cv.phat0$dev)])

# error for pruned tree
pruned.pred = predict(prune.phat0, newdata=phatzero.test)
error.pruned = mean((pruned.pred - phatzero.test$redshift)^2)
error.pruned
```

```
## [1] 0.01932991
```

```
# We also see that the pruned tree has a larger test MSE.


# Scale data for neural network method
train.max.scale = apply(phatzero.train, 2, max)
train.min.scale = apply(phatzero.train, 2, min)
train.scaled = as.data.frame(scale(phatzero.train, center=train.min.scale,
                                   scale=train.max.scale - train.min.scale))

test.max.scale = apply(phatzero.test, 2, max)
test.min.scale = apply(phatzero.test, 2, min)
test.scaled = as.data.frame(scale(phatzero.test, center=test.min.scale,
                                  scale=test.max.scale - test.min.scale))


# First we try a simple single hidden-layer (with 5 neurons) model together
```

```
# with the logistic activation function and back-propogation algorithm.

# Create the formula for the model
n = names(train.scaled)
formula.nn = as.formula(paste("redshift ~", paste(n[!n %in% "redshift"], collapse = "+")))
# Fit the model on the training data.

set.seed(3)
NN0 = neuralnet(formula = formula.nn, data = train.scaled, hidden = 5, threshold = 0.1,
                stepmax = 1e5, act.fct = "logistic", linear.output=T)
nnet.fit.err <- NN0$result.matrix[1,1]
nnet.fit.err
```

```
## [1] 1.604277939
```

```
# The error in the result summary is about 1.60 which is low. However, when we
# compute the training error below manually we get a much smaller number, 0.0013

# train error
train.fitted.values <- neuralnet::compute(NN0, train.scaled[,2:12])
trainerrorNN0 <- mean((train.scaled$redshift - train.fitted.values$net.result)^2)
trainerrorNN0
```

```
## [1] 0.001283422351
```

```
# The train error is about 0.0013 ##

# We now obtain the test error ##
# Get the predicted values on the test set and scale them back.
pred.NN0.scaled = neuralnet::compute(NN0, test.scaled[,2:12])
# Scale back ##
NN0.preds.scaledback = pred.NN0.scaled$net.result*(max(phatzero.test$redshift) -
                                                    min(phatzero.test$redshift))+
  min(phatzero.test$redshift)

# Compute the test MSE
NN0.test.mse = mean((phatzero.test$redshift - NN0.preds.scaledback)^2)
NN0.test.mse
```

```
## [1] 0.0152825406
```

```
# We see that the test error is about 0.015, which is quite low.

# We now make the network slightly more complex by adding a second layer
# containing 3 nodes and using the "tanh" activation function which in this
# case gives lower error than when we use the "logistic" function, other things being equal.

set.seed(3)
NN1 = neuralnet(formula = formula.nn, data = train.scaled, hidden = c(5,3),
                threshold = 0.1, stepmax = 1e5, act.fct = "tanh", linear.output=TRUE)

NN1.fit.err <- NN1$result.matrix[1,1]
NN1.fit.err
```

```
## [1] 0.8905823958
```

```r
# The fit error given by the result summary is about 0.89.

# train error
train.fitted.values <- neuralnet::compute(NN1, train.scaled[,2:12])
trainerrorNN1 <- mean((train.scaled$redshift - train.fitted.values$net.result)^2)
trainerrorNN1
```

```
## [1] 0.0007124659167
```

```r
# The train error is very low at about 0.0007.

# We now obtain the test error
# Get the predicted values on the test set and scale them back.
pred.NN1.scaled = neuralnet::compute(NN1, test.scaled[,2:12])
# Scale back
NN1.preds.scaledback = pred.NN1.scaled$net.result*(max(phatzero.test$redshift) -
                                                    min(phatzero.test$redshift)) +
  min(phatzero.test$redshift)

# Compute the test MSE
NN1.test.mse = mean((phatzero.test$redshift - NN1.preds.scaledback)^2)
NN1.test.mse
```

```
## [1] 0.0139118008
```

```r
# We see that the test MSE is 0.014 in this case, which is slightly lower than the
# previous simpler network.

# Lastly, we try yet another more complex network consisting of three layers, of 7,5,3 nodes,
# respectively, using the "tanh" activation function.


set.seed(3)
NN2 = neuralnet(formula = formula.nn, data = train.scaled, hidden = c(7,5,3),
                threshold = 0.1, stepmax = 1e5, act.fct = "tanh", linear.output=TRUE)

NN2.fit.err <- NN2$result.matrix[1,1]
NN2.fit.err
```

```
## [1] 0.5725076711
```

```r
# The fit error given by the result summary is about 0.6, lower than the previous two
# configurations (and lower than when we use the "logistic" activation function)

## train error
train.fitted.values <- neuralnet::compute(NN2, train.scaled[,2:12])
trainerrorNN2 <- mean((train.scaled$redshift - train.fitted.values$net.result)^2)
trainerrorNN2
```

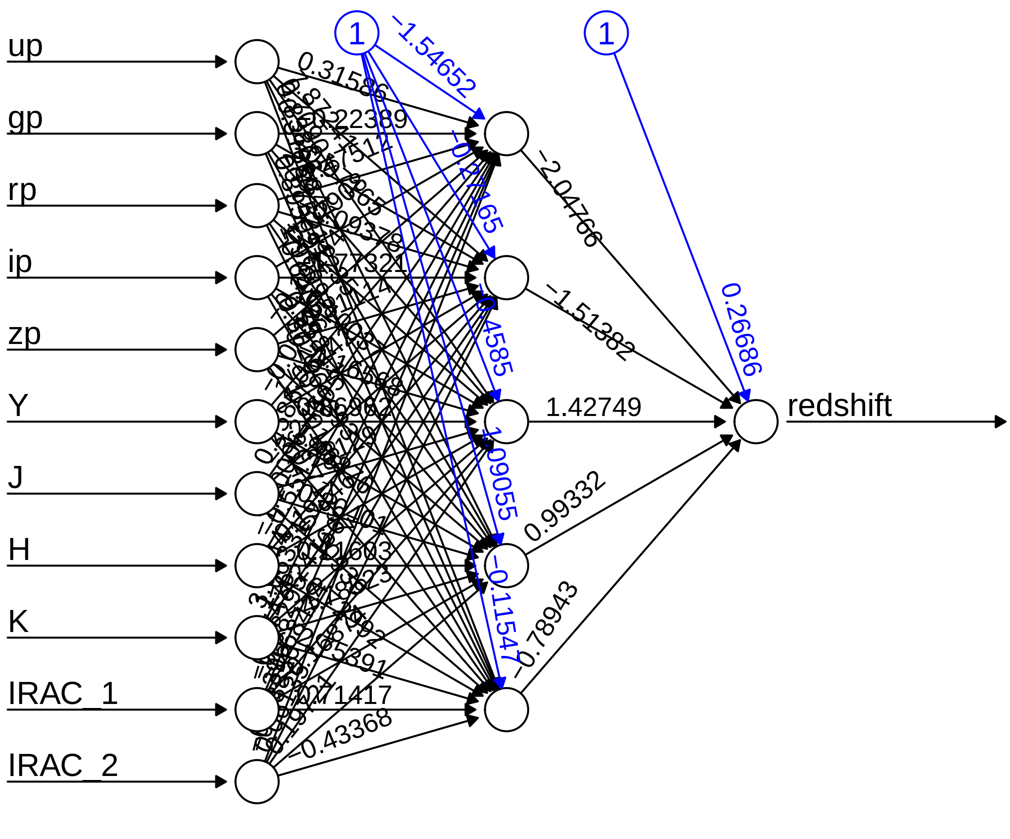
```
## [1] 0.0004580061369
```

```r
# The manually computed train error is about 0.0004, also lower than the previous two networks.

# We now obtain the test error
# Get the predicted values on the test set and scale them back
pred.NN2.scaled = neuralnet::compute(NN2, test.scaled[,2:12])
# Scale back
```

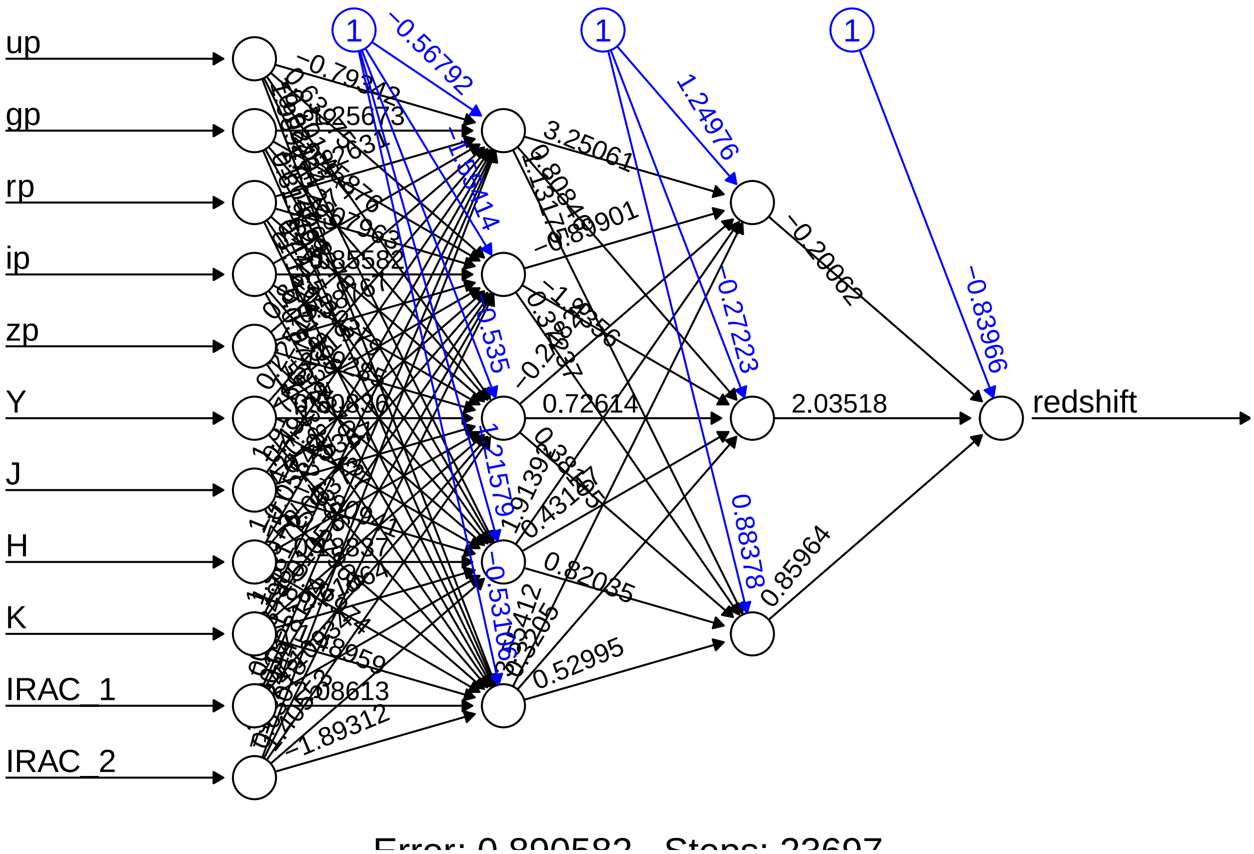
```r
NN2.preds.scaledback = pred.NN2.scaled$net.result*(max(phatzero.test$redshift) -
                                                  min(phatzero.test$redshift)) +
  min(phatzero.test$redshift)

# Compute the test MSE
NN2.test.mse = mean((phatzero.test$redshift - NN2.preds.scaledback)^2)
NN2.test.mse
```

```
## [1] 0.04564603961
```

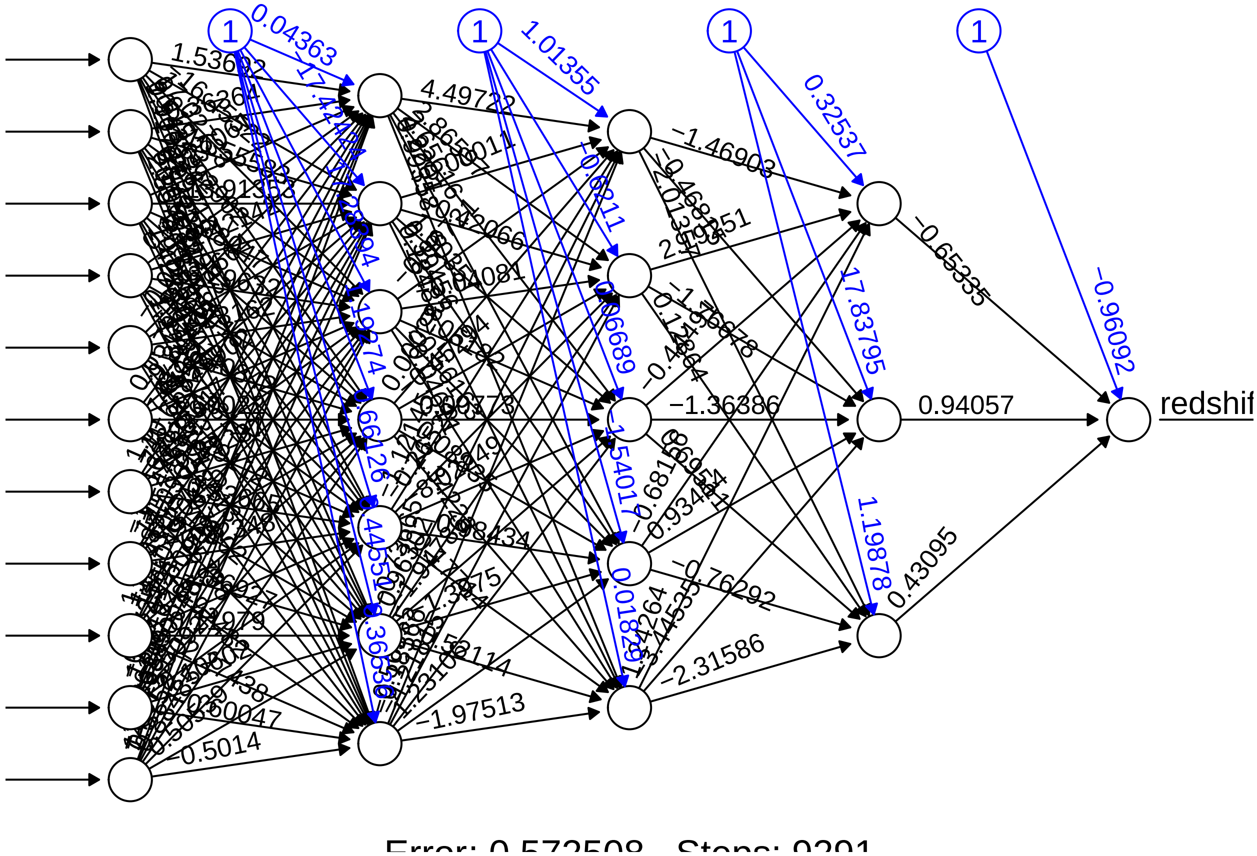
```r
# We see that the test MSE for this more complicated model is 0.04, which is slightly higher
# than the previous two simpler networks.

# plot the netwrosk
par(mfrow=c(3,1))
plot(NN0, rep="best")
```




```r
plot(NN1, rep="best")
```

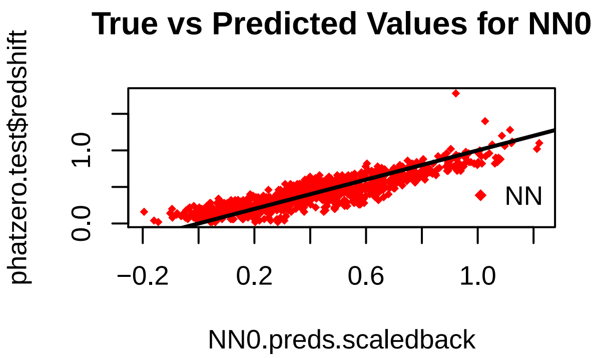


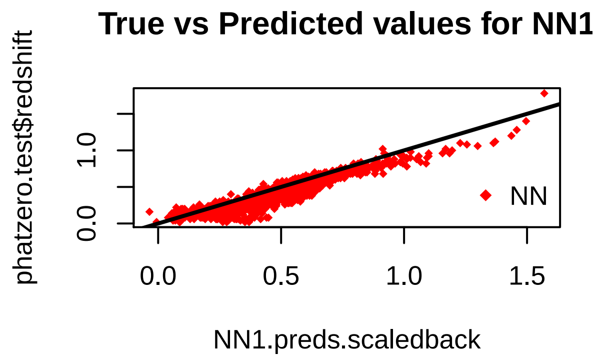

```r
plot(NN2, rep="best")
```

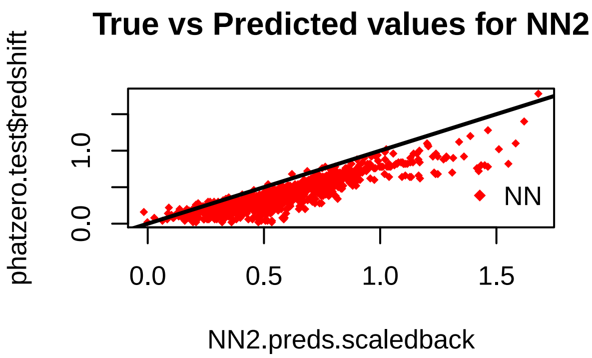



```r
# Plot the three neural networks against the observed test responses to asses

par(mfrow=c(2,2))
plot(NN0.preds.scaledback, phatzero.test$redshift, col='red',
     main='True vs Predicted Values for NN0',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red', bty='n')


plot(NN1.preds.scaledback, phatzero.test$redshift ,col='red',
     main='True vs Predicted values for NN1',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red', bty='n')


plot(NN2.preds.scaledback, phatzero.test$redshift,col='red',
     main='True vs Predicted values for NN2',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red', bty='n')

# We see that the first two plots corresponding to the first and second networks with
# very low test MSE's indicate that the predicted values are close to actual values.

# Comparing all methods via a boxplot of the errors
error.all = c(error.lasso, error.pcr, error.pls, error.ridge,
              error.unpruned, error.pruned, NN1.test.mse)
```

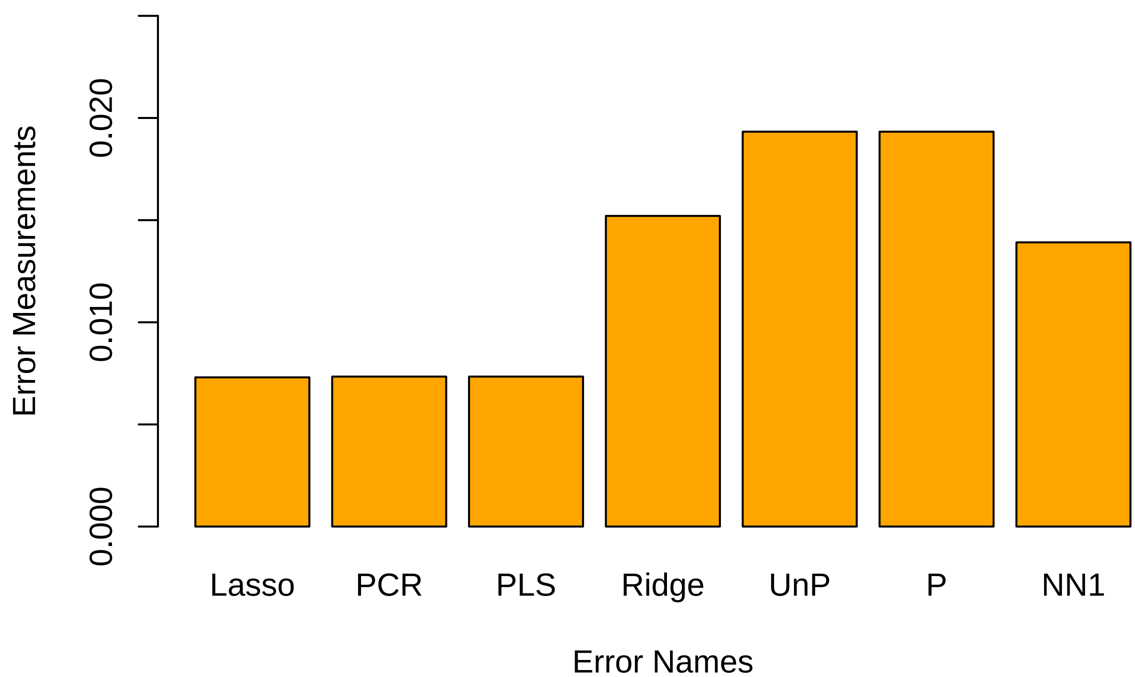
```r
names(error.all) = c("Lasso", "PCR", "PLS", "Ridge", "UnP", "P", "NN1")
par(mfrow=c(1,1))
```








```r
barplot(error.all, col="orange" ,xlab="Error Names", ylab="Error Measurements", ylim = c(0,0.025))
```




```r
which.min(error.all)
```

```
## Lasso
```

```
## 1
```

```r
# We see that the Lasso, PCR, and PLS models produce the lowest test MSEs, all
# very close, with Lasso being slightly lower than the other two, and so the lowest
# in all models. The neural network with 2 hidden layers and the "tanh" activation function
# produces the next lowest test MSE, and so outperforms Ridge and tree regressions.
```

“‘